Domain-Specific Visualization of Alloy Instances

Loïc Gammaitoni and Pierre Kelsen

University of Luxembourg

Motivation Alloy is a modelling language based on first order logic and relational calculus combining the precision of formal specifications with powerful automatic analysis features [3]. The automatic analysis is done by the Alloy Analyzer, a tool capable of finding satisfiable instances for given Alloy models in a finite domain space using Sat-solving.

The utility of this instance finding mechanisms relies on the small scope hypothesis that claims that most design errors can be detected in small model instances. Those design errors can be identified by evaluating expressions in the instances generated, but also through the direct visualization of those instances. In the latter case, the intuitiveness and expressiveness of the instance visualization play a key role to efficiently diagnose the validity of instance.

The current visualizations offered by the Alloy Analyzer produce visual diagrams that are close to the structure of instances. These low level visualizations become difficult to read when instances become more complex.

To highlight this lack of intuitivity, let us consider the formalisation of a Finite State Machine (FSM). Figure 1 is the visualization of a FSM instance provided by the Alloy Analyzer, while fig. 2 is an intuitive domain-specific visualization produced by Lightning, a tool implementing the approach described in this paper.

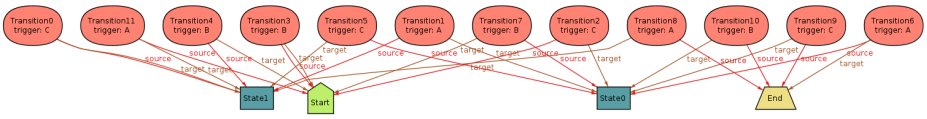

Fig. 1. Visualization of an FSM instance as given by the Alloy Analyzer

Approach Overview In order to clearly frame the set of possible visualizations definable using our approach, we designed a Visual Language Model (VLM) as generic as possible in Alloy, such that a large variety of domain specific visualizations can be supported. Works in [2] provide notions on how such a visual language can be defined. This VLM contains basic visual elements such as shapes and texts which can be composed or related to each other via connectors and can be arranged through the definition of layouts.

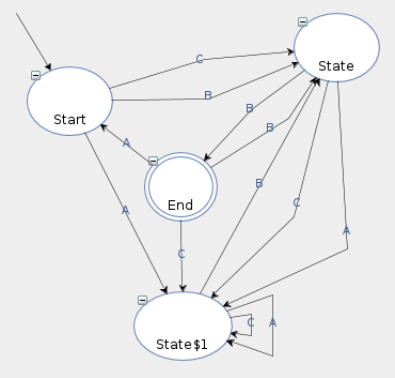

Fig. 2. Instance depicted in fig. 1 visualized using Lightning

The visualization of instances for a given model m is defined as a model transformation from m to VLM.

We define this transformation through a set of mappings and set of constraints, all expressed in Alloy. The constraints enforce that any instance of the transformation contains an instance of m as well as its corresponding VLM instance.

The VLM instance contained in the transformation instance can then be interpreted and rendered accordingly.

In fig. 3 we provide an overview of this approach. We call m2VLM the Alloy module containing the transformation specifications.

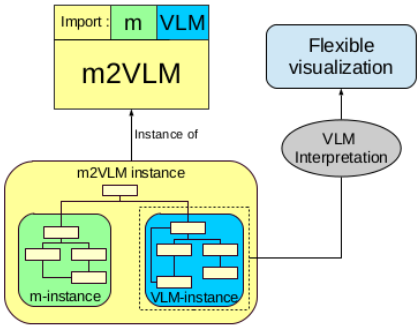

Fig. 3. overview of our approach

This whole process is implemented by the Lightning tool[4].

Transformations A transformation is composed of backbone mappings and integration constraints. Each mapping composing the backbone of the transformation is expressed via binary relations as illustrated in fig. 4.

```
one sig Bridge{
   map1: State one -> lone ELLIPSE,
   map2: End one -> one DOUBLE_ELLIPSE,
   map3: Transition lone -> one CONNECTOR,
   map4: Start lone -> one CONNECTOR,
   map5: State one -> one TEXT
}
```

Fig. 4. Backbone mappings of the FSM2VLM transformation

Guard predicates are used to define a condition under which a given element of the input model belongs to a backbone mapping (fig. 5).

```
  pred in_map1(s:State){
// all the state except the end state are to be represented by an ellipse (map1)
      s in State - End
  }
```

Fig. 5. Example of guard present in the FSM2VLM transformation

The way the image of a given element is integrated in the result of the transformation is defined through integration constraints. Those constraints are rules which have the particularity to be interpretable (they can be seen as imperative assignments or calls, rather than declarative constraints). We give example of such rules in fig.6

```
 pred prop_map4(t:Transition ,  c:CONNECTOR){
// color of the connector is black
     c.color=BLACK
// connector source point to the visual element representing t.source
     c.source=Bridge.(map1+map2)[t.source]
// connector target point to the visual element representing t.target
     c.target= Bridge.(map1+map2)[t.target]
// the connector is labelled according to the trigger of the transition.
     c.connectorLabel[0]=t.trigger
 }
```

Fig. 6. Example of rules present in the FSM2VLM transformation

From Analysis to Interpretation We can use the Alloy Analyzer to generate the VLM instance from a given m-instance. This is done by promoting a given m-instance into an Alloy model and then solving the transformation model together with this promoted instance. Unfortunately this approach is not really practical because the analysis itself may be quite time consuming, being based on SAT solving.

In recent work [1], we discuss the use of functional Alloy modules to improve the performance of the Alloy analysis of functional transformations. As it is reasonable to assume that each instance of a given module m should have exactly one corresponding visualization, we can consider the transformation defined to achieve such a visualization as a function. Visualizations can thus be defined by a functional module.

Besides making the computation of the visualization really fast, the use of functional modules relieves us of another limitation induced of the Alloy analyzer: scope calculation. The determination of a small scope is important in order to limit the running time of the analyzer. Finding optimal scopes is in general an decidable problem. Interpreting Alloy modules via functional modules rather than analyzing them makes scope determination unnecessary.

Concluding Discussion In this paper, we presented an approach to define intuitive visualizations for Alloy instances. The approach uses Alloy itself to define a visual model and a transformation from the input instance to this visual instance. We use the recently introduced notion of functional modules to make our approach practical: rather than doing SAT solving to obtain the visual instance, we interpret the input instance and the transformation specification to directly compute the visual instance.

In the future, we plan further investigation on the use of Alloy to define different aspects of modelling languages (semantics, concrete syntax[1],...), as well as combinations of those aspects (visualization of semantics,...) The ultimate goal is to develop a language workbench based on Alloy.

References

1. Loïc Gammaitoni and Pierre Kelsen. Functional Alloy Modules. Submitted as research paper to ABZ 2014.
2. Sergio Orefice Gennaro Costagliola, Andrea De Lucia and Giuseppe Polese. A classification framework to support the design of visual languages. *Journal of Visual Languages and Computing*, pages 573–600, 2002.
3. Daniel Jackson. *Software abstractions*. MIT press Cambridge, 2012.
4. Luxembourg University. Lightning tool-website, http://lightning.gforge.uni.lu.

[1] A full concrete syntax support for a given model generally implies the ability to create or modify instances of this model through their visualizations. The visualization approach introduced in this paper is a first step towards a full concrete syntax support